Aufgabe 1

Ändern Sie die Sprache **WHILE** ab, indem Sie anstelle des atomaren Ausdruckes `read` Anweisungen der Form `read I` zulassen. Die Semantik dieser Anweisung lautet informell: Die Ausführung von `read I` bewirkt eine Zuweisung des nächsten Eingabewertes an die Variable `I` und eine Verkürzung der Eingabedatei um das erste Element.
Formalisieren Sie die Semantik von `read I` denotationell.

$$\mathcal{C}[\![\underline{read}\ I]\!]\ (s,e,a) = \begin{cases} (S[n/I], e', a), & \text{falls } \mathcal{T}[\![\underline{read}]\!]\ s\ e = (n, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

Soll heißen: der nächste gelesene Eingabewert (n) ersetzt (I) im Speicher. (e') ist ein um den ersten Wert kürzeres (e).

Aufgabe 2

Erweitern Sie die Sprache **WHILE** um Anweisungen der Form

```
for I := T_1 to T_2 do C
```

. Formalisieren Sie die Semantik dieser Anweisungen denotationell.

Diese for-Schleife lässt einem viel Raum zur Interpretation. Zum Beispiel trifft der Ausdruck $I := T_1\ \underline{to}\ T_2$ keine Aussage darüber, wie T_1 und T_2 miteinander verglichen werden. Ebenfalls ist unklar, ob T_1 nach, vor, oder während der Ausführung von C inkrementiert, dekrementiert, o. ä., oder zugewiesen wird. Wir gehen davon aus, das folgende denotationelle Semantik - angelehnt an die While-Definition - ausreichend ist.

$$\mathcal{C}[\![\underline{for}\ I := T_1\ \underline{to}\ T_2\ \underline{do}\ C]\!]\ (s,e,a) = \begin{cases} \mathcal{C}[\![C; \underline{for}\ I\ \underline{to}\ T_2\ \underline{do}\ C]\!](s,e',a), & \text{falls} \\ \qquad \mathcal{B}[\![T_1\ \underline{BOP}\ T_2]\!]\ s\ e = (\underline{false}, e') \\ (s,e',a), & \text{falls} \quad \mathcal{B}[\![T_1\ \underline{BOP}\ T_2]\!]\ s\ e = (\underline{true}, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

Anmerkung: Die erste Zuweisung von $I := T_1$ macht das Gesamte Konstrukt eigentlich noch komplizierter. Besser wäre eine Semantik mit einer weiteren Fallunterscheidung. Man hätte dann folgende Fälle: erste Ausführung, alle weiteren Ausführungen bei denen die Auswertung $\underline{false}$ ergibt, den Fall wo die logische Operation $\underline{true}$ ergibt und den Fehlerfall.

Aufgabe 3

Erweitern Sie die Sprache WHILE um den atomaren booleschen Term `eof`. Die informelle Semantik von `eof` lautet: `eof` ist wahr gdw die Eingabe leer ist.
Formalisieren Sie die Semantik von `eof` denotationell.

$$B[\![\text{eof}]\!]z = \begin{cases} (\text{wahr}, z') & \text{falls } e = \epsilon \\ (\text{falsch}, z') & \text{sonst} \end{cases}$$

Aufgabe 4

Programmieren Sie in WHILE (einschließlich `eof`) einen Algorithmus zur Berechnung der Summe aller Eingabewerte. Beweisen Sie die Korrektheit Ihres Programms anhand der denotationellen Semantik. Diskutieren Sie die Problematik beim Fehlen von `eof`.
Programm in `WHILE`:

```
zahl := read
while ¬ eof do zahl := zahl + read
```

Diskussion:

Ohne `eof` bräuchte das Programm für das Einlesen der kompletten Datei mittels einer Schleife eine Fehlerbehandlung (Exception-Handling), um eine leere Eingabe abzufangen, da das Programm sonst abstürzen würde (`read` wirft einen Fehler bei leerer Eingabe).

Beweisskizze: (Strukturelle Induktion)

Induktionsanfang: Leere Datei. Das Programm wird korrekt ausgeführt, die Bedingung der While-Schleife wird als falsch ausgewertet.

Induktionsschritt: Datei mit beliebig vielen Zahlen wird solange ausgewertet, bis die Datei leer ist. Die Schleife bricht wie im Induktionsanfang beschrieben.

$$\left.\begin{array}{l} x := 0 \ \} C_1 \\ \left.\begin{array}{l} \left.\begin{array}{l} \underline{while}\ \neg \underline{eof}\ \underline{do} \\ \quad x := x + \underline{read}\ \} C_5 \end{array}\right\} C_2 \\ \underline{output}\ x\ \} C_3 \end{array}\right\} C_4 \end{array}\right\} i$$

$$\mathcal{P}[C'](n_1,\dots,n_k) = (\sum_{i=1}^{k} n_i)$$

$$\Leftarrow \mathcal{C}[C]\underbrace{(s_0,(n_0,...,n_k),\epsilon}_{z} \qquad = (s,\epsilon,(\sum_{i=1}^{k} n_i))$$

$$\mathcal{C}[C_1,C_4]z = \mathcal{C}[C_4](\mathcal{C}[x := 0]z)$$

$$= \mathcal{C}[C_4]\underbrace{(s_0[0/x],(n_1,...,n_k),\epsilon)}_{z_0}$$

sei

$$z := (s_0[(\sum_{i=1}^{i} n_j/x)],(n_{i+1},...,n_k),\epsilon) = \mathcal{C}[C_2,C_3]z_0 = \mathcal{C}[C_3](\mathcal{C}[C_2]z_0)$$

Beh.:

$$\mathcal{C}[C_2]z_i = \begin{cases} \mathcal{C}[C_2]z_{i+1} & \text{falls } i < k \\ z_k & \text{falls } i = k \end{cases}$$

$$\mathcal{C}[\underline{while}\ \neg\underline{eof}\ \underline{do}\ C_5]z_i = \mathcal{C}[C_2](\mathcal{C}[C_5]z_i)$$

$$\mathcal{C}[x := x + read]z_i = z_{i+1}$$